

Bayesian Deep Learning

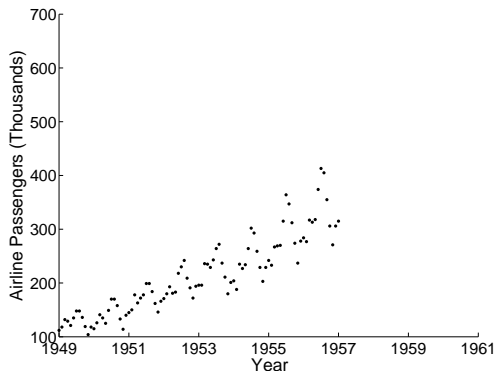
Andrew Gordon Wilson

Assistant Professor

<https://people.orie.cornell.edu/andrew>
Cornell University

Toronto Deep Learning Summer School
July 31, 2018

Model Selection



Which model should we choose?

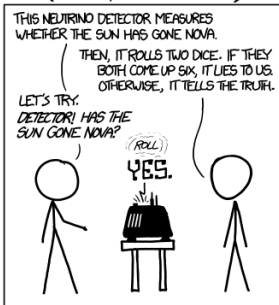
(1): $f_1(x) = a_0 + a_1x$

(2): $f_2(x) = \sum_{j=0}^3 a_jx^j$

(3): $f_3(x) = \sum_{j=0}^{10^4} a_jx^j$

Bayesian or Frequentist?

DID THE SUN JUST EXPLODE?
(IT'S NIGHT, SO WE'RE NOT SURE.)



FREQUENTIST STATISTICIAN:

THE PROBABILITY OF THIS RESULT HAPPENING BY CHANCE IS $\frac{1}{36} = 0.027$.
SINCE $p < 0.05$, I CONCLUDE THAT THE SUN HAS EXPLODED.

A stick figure stands to the left of the detector, looking at it.

BAYESIAN STATISTICIAN:

BET YOU \$50 IT HASN'T.

A stick figure stands to the right of the detector, looking at it.

Bayesian Deep Learning

Why?

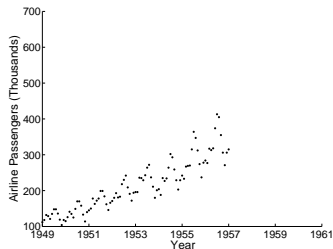
- ▶ A powerful framework for model construction and understanding generalization
- ▶ Uncertainty representation (crucial for decision making)
- ▶ *Better point estimates*
- ▶ It was the most successful approach at the end of the second wave of neural networks (Neal, 1998).
- ▶ Neural nets are much less mysterious when viewed through the lens of probability theory.

Why not?

- ▶ Can be computationally intractable (but doesn't have to be).
- ▶ Can involve a lot of moving parts (but doesn't have to).

There has been exciting progress in the last two years addressing these limitations as part of an extremely fruitful research direction.

How do we build models that learn and generalize?



Basic Regression Problem

- ▶ Training set of N targets (observations) $\mathbf{y} = (y(x_1), \dots, y(x_N))^T$.
- ▶ Observations evaluated at inputs $X = (x_1, \dots, x_N)^T$.
- ▶ Want to predict the value of $y(x_*)$ at a test input x_* .

For example: Given CO_2 concentrations \mathbf{y} measured at times X , what will the CO_2 concentration be for $x_* = 2024$, 10 years from now?

Just knowing high school math, what might you try?

How do we build models that learn and generalize?

Guess the parametric form of a function that could fit the data

- ▶ $f(x, \mathbf{w}) = \mathbf{w}^T x$ [Linear function of \mathbf{w} and x]
- ▶ $f(x, \mathbf{w}) = \mathbf{w}^T \phi(x)$ [Linear function of \mathbf{w}] (Linear Basis Function Model)
- ▶ $f(x, \mathbf{w}) = g(\mathbf{w}^T \phi(x))$ [Non-linear in x and \mathbf{w}] (E.g., Neural Network)

$\phi(x)$ is a vector of basis functions. For example, if $\phi(x) = (1, x, x^2)$ and $x \in \mathbb{R}^1$ then $f(x, \mathbf{w}) = w_0 + w_1x + w_2x^2$ is a quadratic function.

Choose an error measure $E(\mathbf{w})$, minimize with respect to \mathbf{w}

- ▶ $E(\mathbf{w}) = \sum_{i=1}^N [f(x_i, \mathbf{w}) - y(x_i)]^2$

How do we build models that learn and generalize?

A probabilistic approach

We could explicitly account for noise in our model.

- ▶ $y(x) = f(x, \mathbf{w}) + \epsilon(x)$, where $\epsilon(x)$ is a noise function.

One commonly takes $\epsilon(x) = \mathcal{N}(0, \sigma^2)$ for i.i.d. additive Gaussian noise, in which case

$$p(y(x)|x, \mathbf{w}, \sigma^2) = \mathcal{N}(y(x); f(x, \mathbf{w}), \sigma^2) \quad \text{Observation Model} \quad (1)$$

$$p(\mathbf{y}|x, \mathbf{w}, \sigma^2) = \prod_{i=1}^N \mathcal{N}(y(x_i); f(x_i, \mathbf{w}), \sigma^2) \quad \text{Likelihood} \quad (2)$$

- ▶ Maximize the likelihood of the data $p(\mathbf{y}|x, \mathbf{w}, \sigma^2)$ with respect to σ^2, \mathbf{w} .

For a Gaussian noise model, this approach will make the same predictions as using a squared loss error function:

$$\log p(\mathbf{y}|X, \mathbf{w}, \sigma^2) \propto -\frac{1}{2\sigma^2} \sum_{i=1}^N [f(x_i, \mathbf{w}) - y(x_i)]^2 \quad (3)$$

How do we build models that learn and generalize?

- ▶ The probabilistic approach helps us interpret the error measure in a deterministic approach, and gives us a sense of the noise level σ^2 .
- ▶ Both approaches are prone to *over-fitting* for flexible $f(x, \mathbf{w})$: low error on the training data, high error on the test set.

Regularization

- ▶ Use a penalized log likelihood (or error function), such as

$$E(\mathbf{w}) = \underbrace{-\frac{1}{2\sigma^2} \sum_{i=1}^n (f(x_i, \mathbf{w}) - y(x_i))^2}_{\text{model fit}} \underbrace{-\lambda \mathbf{w}^T \mathbf{w}}_{\text{complexity penalty}}. \quad (4)$$

- ▶ **But how should we define and penalize complexity?**
- ▶ Can set λ using *cross-validation*.
- ▶ Same as maximizing a posterior $\log p(\mathbf{w}|\mathbf{y}, X) = \log p(\mathbf{y}|\mathbf{w}, X) + p(\mathbf{w})$ with a Gaussian prior $p(\mathbf{w})$. **But this is not Bayesian!**

Bayesian Inference

Bayes' Rule

$$p(a|b) = p(b|a)p(a)/p(b), \quad p(a|b) \propto p(b|a)p(a). \quad (5)$$

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{marginal likelihood}}, \quad p(\mathbf{w}|\mathbf{y}, X, \sigma^2) = \frac{p(\mathbf{y}|X, \mathbf{w}, \sigma^2)p(\mathbf{w})}{p(\mathbf{y}|X, \sigma^2)}. \quad (6)$$

Sum Rule

$$p(x) = \sum_y p(x, y) \quad (7)$$

Product Rule

$$p(x, y) = p(x|y)p(y) = p(y|x)p(x) \quad (8)$$

Predictive Distribution

Sum rule: $p(x) = \sum_y p(x, y)$. **Product rule:** $p(x, y) = p(x|y)p(y) = p(y|x)p(x)$.

$$p(y|x_*, \mathbf{y}, X) = \int p(y|x_*, \mathbf{w})p(\mathbf{w}|\mathbf{y}, X)d\mathbf{w}. \quad (9)$$

- ▶ Think of each setting of \mathbf{w} as a different model. Eq. (9) is a *Bayesian model average*, an average of infinitely many models weighted by their posterior probabilities.
- ▶ No over-fitting, automatically calibrated complexity.
- ▶ **Eq. (9) is not analytic for many likelihoods $p(\mathbf{y}|X, \mathbf{w}, \sigma^2)$ and priors $p(\mathbf{w})$.** (But recent advances such as SG-HMC have made these computations much more tractable in deep learning).
- ▶ Typically more interested in the induced distribution over **functions** than in parameters \mathbf{w} . Can be hard to have intuitions for priors on $p(\mathbf{w})$.

Example: Bent Coin

Suppose we flip a bent coin with probability λ of landing tails.

1. What is the likelihood of a set of data

$$\mathcal{D} = \{x_1, x_2, \dots, x_n\}?$$

2. What is the maximum likelihood solution for λ ?
3. Suppose we observe 2 tails in the first two flips. What is the probability that the next flip will be a tails, using maximum likelihood?

Example: Bent Coin

Likelihood of getting m tails is

$$p(\mathcal{D}|m, \lambda) = \binom{N}{m} \lambda^m (1 - \lambda)^{N-m} \quad (10)$$

If we choose a prior $p(\lambda) \propto \lambda^\alpha (1 - \lambda)^\beta$ then the posterior will have the same functional form as the prior.

Example: Bent Coin

Likelihood of getting m tails is

$$p(\mathcal{D}|m, \lambda) = \binom{N}{m} \lambda^m (1 - \lambda)^{N-m} \quad (11)$$

If we choose a prior $p(\lambda) \propto \lambda^\alpha (1 - \lambda)^\beta$ then the posterior will have the same functional form as the prior.

We can choose the beta distribution:

$$\text{Beta}(\lambda|a, b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \lambda^{a-1} (1 - \lambda)^{b-1} \quad (12)$$

The Gamma functions ensure that the distribution is normalised:

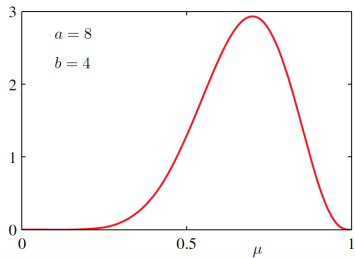
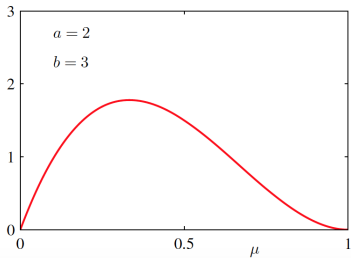
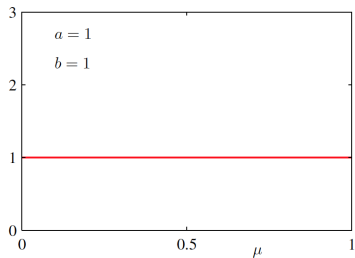
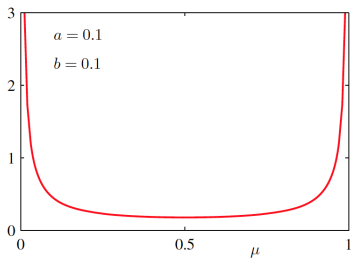
$$\int \text{Beta}(\lambda|a, b) d\lambda = 1 \quad (13)$$

Moments:

$$\mathbb{E}[\lambda] = \frac{a}{a+b} \quad (14)$$

$$\text{var}[\lambda] = \frac{ab}{(a+b)^2(a+b-1)}. \quad (15)$$

Beta Distribution



Example: Bent Coin

Applying Bayes theorem, we find:

$$p(\lambda|\mathcal{D}) \propto p(\mathcal{D}|\lambda)p(\lambda) \quad (16)$$

$$= \text{Beta}(\lambda; m + a, N - m + b) \quad (17)$$

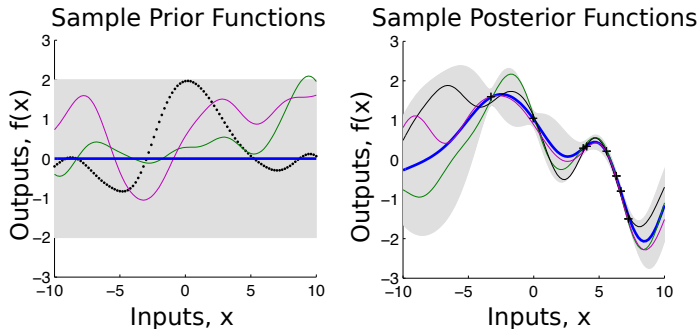
We can view a and b as pseudo-observations!

$$\mathbb{E}[\lambda|\mathcal{D}] = \frac{m + a}{N + a + b} \quad (18)$$

1. What is the probability that the next flip is tails?
2. What happens in the limits of a, b ?
3. What happens in the limit of infinite data?

A Function Space View: Gaussian processes

$$\overbrace{p(f(x)|\mathcal{D})}^{\text{Function posterior}} \propto \overbrace{p(\mathcal{D}|f(x))}^{\text{Likelihood}} \overbrace{p(f(x))}^{\text{Function prior}}$$



Radford Neal showed in 1996 that a Bayesian neural network with an *infinite* number of hidden units is a Gaussian process!

Gaussian processes

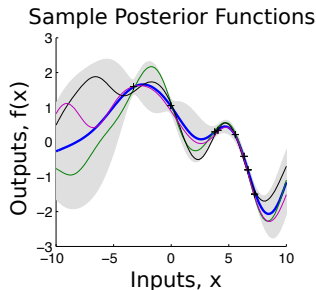
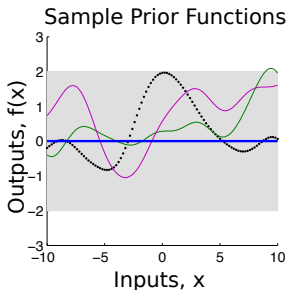
Definition

A Gaussian process (GP) is a collection of random variables, any finite number of which have a joint Gaussian distribution.

Nonparametric Regression Model

- Prior: $f(x) \sim \mathcal{GP}(m(x), k(x, x'))$, meaning $(f(x_1), \dots, f(x_N)) \sim \mathcal{N}(\boldsymbol{\mu}, K)$, with $\boldsymbol{\mu}_i = m(x_i)$ and $K_{ij} = \text{cov}(f(x_i), f(x_j)) = k(x_i, x_j)$.

$$\underbrace{p(f(x)|\mathcal{D})}_{\text{GP posterior}} \propto \underbrace{p(\mathcal{D}|f(x))}_{\text{Likelihood}} \underbrace{p(f(x))}_{\text{GP prior}}$$

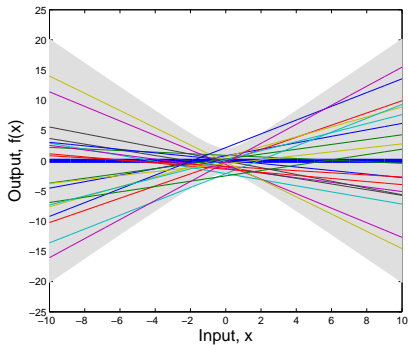


Linear Basis Models

Consider the simple linear model,

$$f(x) = a_0 + a_1x, \quad (19)$$

$$a_0, a_1 \sim \mathcal{N}(0, 1). \quad (20)$$

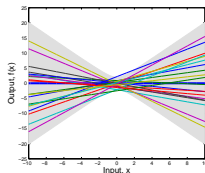


Linear Basis Models

Consider the simple linear model,

$$f(x) = a_0 + a_1x, \quad (21)$$

$$a_0, a_1 \sim \mathcal{N}(0, 1). \quad (22)$$



Any collection of values has a joint Gaussian distribution

$$[f(x_1), \dots, f(x_N)] \sim \mathcal{N}(0, K), \quad (23)$$

$$K_{ij} = \text{cov}(f(x_i), f(x_j)) = k(x_i, x_j) = 1 + x_i x_j. \quad (24)$$

By definition, $f(x)$ is a Gaussian process.

Linear Basis Function Models

Model Specification

$$f(x, \mathbf{w}) = \mathbf{w}^T \boldsymbol{\phi}(x) \quad (25)$$

$$p(\mathbf{w}) = \mathcal{N}(0, \Sigma_w) \quad (26)$$

Moments of Induced Distribution over Functions

$$\mathbb{E}[f(x, \mathbf{w})] = m(x) = \mathbb{E}[\mathbf{w}^T] \boldsymbol{\phi}(x) = 0 \quad (27)$$

$$\text{cov}(f(x_i), f(x_j)) = k(x_i, x_j) = \mathbb{E}[f(x_i)f(x_j)] - \mathbb{E}[f(x_i)]\mathbb{E}[f(x_j)] \quad (28)$$

$$= \boldsymbol{\phi}(x_i)^T \mathbb{E}[\mathbf{w}\mathbf{w}^T] \boldsymbol{\phi}(x_j) - 0 \quad (29)$$

$$= \boldsymbol{\phi}(x_i)^T \Sigma_w \boldsymbol{\phi}(x_j) \quad (30)$$

- ▶ $f(x, \mathbf{w})$ is a Gaussian process, $f(x) \sim \mathcal{N}(m, k)$ with mean function $m(x) = 0$ and covariance kernel $k(x_a, x_b) = \boldsymbol{\phi}(x_a)^T \Sigma_w \boldsymbol{\phi}(x_b)$.
- ▶ The entire basis function model of Eqs. (25) and (26) is encapsulated as a distribution over functions with kernel $k(x, x')$.

Example: RBF Kernel

$$k_{\text{RBF}}(x, x') = \text{cov}(f(x), f(x')) = a^2 \exp\left(-\frac{\|x - x'\|^2}{2\ell^2}\right) \quad (31)$$

- ▶ Far and above the most popular kernel.
- ▶ Expresses the intuition that function values at nearby inputs are more correlated than function values at far away inputs.
- ▶ The kernel *hyperparameters* a and ℓ control amplitudes and wiggleness of these functions.
- ▶ GPs with an RBF kernel have large support and are *universal approximators*.

Sampling from a GP with an RBF Kernel

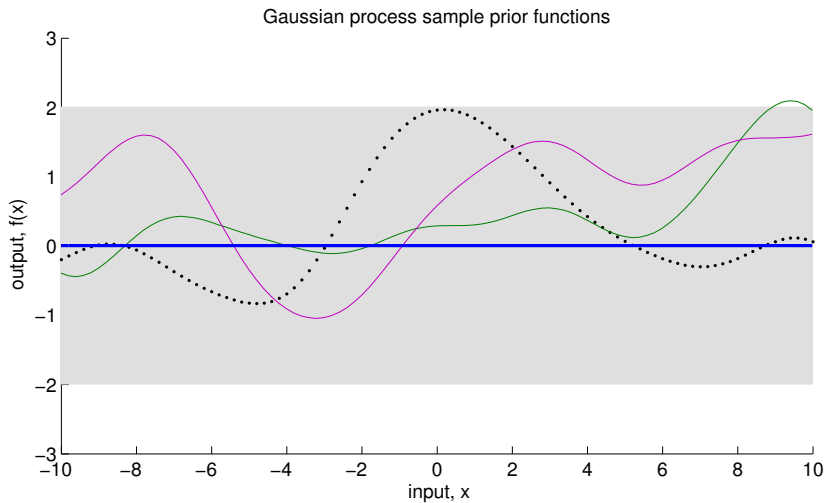
```
x = [-10:0.2:10]'; % inputs (where we query the GP)
N = numel(x); % number of inputs
K = zeros(N,N); % covariance matrix

% very inefficient way of creating K in Matlab
for i=1:N
    for j=1:N
        K(i,j) = k_rbf(x(i),x(j));
    end
end

K = K + 1e-6*eye(N); % add jitter for conditioning of K
CK = chol(K);
f = CK'*randn(N,1); % draws from N(0,K)

plot(x,f);
```

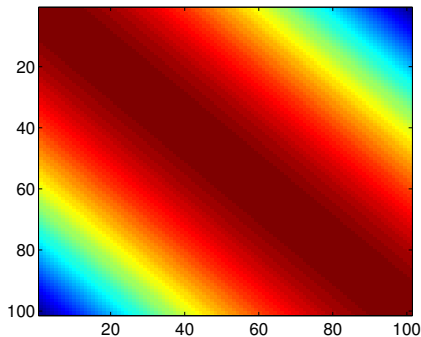
Samples from a GP with an RBF Kernel



RBF Kernel Covariance Matrix

$$k_{\text{RBF}}(x, x') = \text{cov}(f(x), f(x')) = a^2 \exp\left(-\frac{\|x - x'\|^2}{2\ell^2}\right) \quad (32)$$

The covariance matrix K for ordered inputs on a 1D grid. $K_{ij} = k_{\text{RBF}}(x_i, x_j)$.



Learning and Predictions with Gaussian Processes

1. **Learning:** Optimize marginal likelihood,

$$\log p(\mathbf{y}|\boldsymbol{\theta}, X) = \overbrace{-\frac{1}{2}\mathbf{y}^T(\mathbf{K}_\boldsymbol{\theta} + \sigma^2\mathbf{I})^{-1}\mathbf{y}}^{\text{model fit}} - \overbrace{\frac{1}{2}\log|\mathbf{K}_\boldsymbol{\theta} + \sigma^2\mathbf{I}| - \frac{N}{2}\log(2\pi)}^{\text{complexity penalty}},$$

with respect to kernel hyperparameters $\boldsymbol{\theta}$. *The marginal likelihood provides a powerful mechanism for kernel learning.*

2. **Inference:** Conditioned on kernel hyperparameters $\boldsymbol{\theta}$, form the predictive distribution for test inputs X_* :

$$\mathbf{f}_*|X_*, X, \mathbf{y}, \boldsymbol{\theta} \sim \mathcal{N}(\bar{\mathbf{f}}_*, \text{cov}(\mathbf{f}_*)),$$

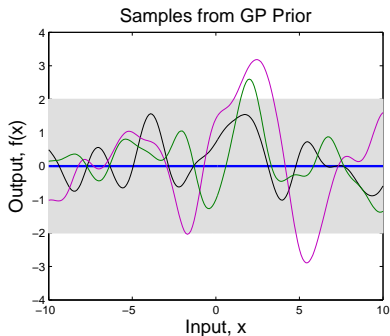
$$\bar{\mathbf{f}}_* = \mathbf{K}_\boldsymbol{\theta}(X_*, X)[\mathbf{K}_\boldsymbol{\theta}(X, X) + \sigma^2\mathbf{I}]^{-1}\mathbf{y},$$

$$\text{cov}(\mathbf{f}_*) = \mathbf{K}_\boldsymbol{\theta}(X_*, X_*) - \mathbf{K}_\boldsymbol{\theta}(X_*, X)[\mathbf{K}_\boldsymbol{\theta}(X, X) + \sigma^2\mathbf{I}]^{-1}\mathbf{K}_\boldsymbol{\theta}(X, X_*).$$

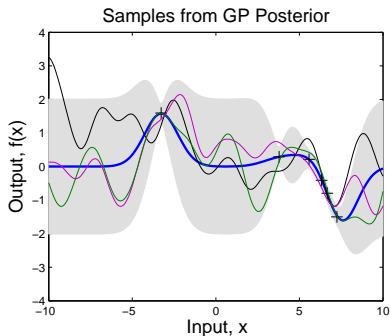
$(\mathbf{K}_\boldsymbol{\theta} + \sigma^2\mathbf{I})^{-1}\mathbf{y}$ and $\log|\mathbf{K}_\boldsymbol{\theta} + \sigma^2\mathbf{I}|$ **naively require $\mathcal{O}(n^3)$ computations, $\mathcal{O}(n^2)$ storage.**

Inference using an RBF kernel

- ▶ Specify $f(x) \sim \mathcal{GP}(0, k)$.
- ▶ Choose $k_{\text{RBF}}(x, x') = a_0^2 \exp(-\frac{\|x-x'\|^2}{2\ell_0^2})$. Choose values for a_0 and ℓ_0 .
- ▶ Observe data, look at the prior and posterior over functions.



(a)

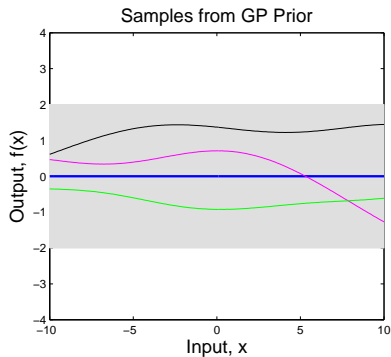


(b)

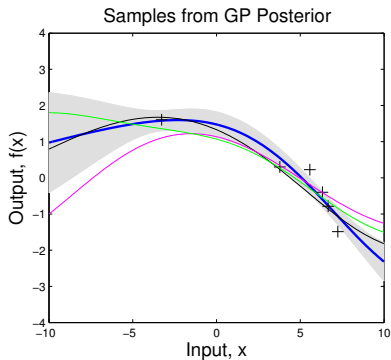
- ▶ Does something look strange about these functions?

Inference using an RBF kernel

Increase the length-scale ℓ .



(a)



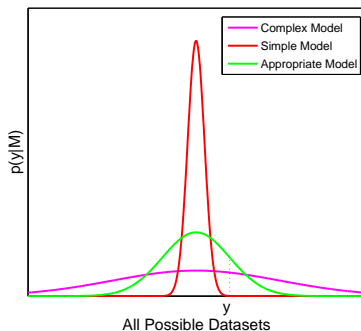
(b)

Learning and Model Selection

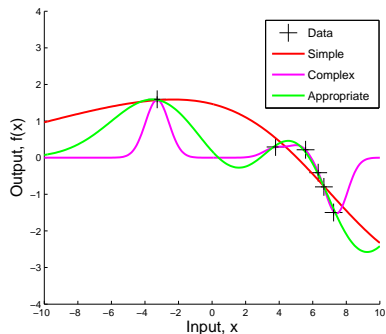
$$p(\mathcal{M}_i|\mathbf{y}) = \frac{p(\mathbf{y}|\mathcal{M}_i)p(\mathcal{M}_i)}{p(\mathbf{y})} \quad (33)$$

We can write the *evidence* of the model as

$$p(\mathbf{y}|\mathcal{M}_i) = \int p(\mathbf{y}|\mathbf{f}, \mathcal{M}_i)p(\mathbf{f})d\mathbf{f}, \quad (34)$$



(a)



(b)

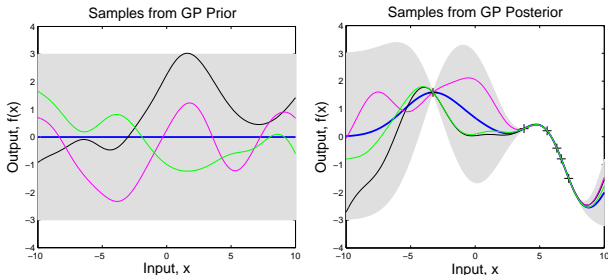
Learning and Model Selection

- ▶ We can integrate away the entire Gaussian process $f(x)$ to obtain the marginal likelihood, as a function of kernel hyperparameters θ alone.

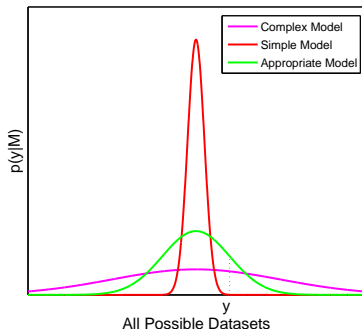
$$p(\mathbf{y}|\boldsymbol{\theta}, X) = \int p(\mathbf{y}|\mathbf{f}, X)p(\mathbf{f}|\boldsymbol{\theta}, X)d\mathbf{f}. \quad (35)$$

$$\log p(\mathbf{y}|\boldsymbol{\theta}, X) = \underbrace{-\frac{1}{2}\mathbf{y}^T(K_{\boldsymbol{\theta}} + \sigma^2 I)^{-1}\mathbf{y}}_{\text{model fit}} - \underbrace{\frac{1}{2}\log |K_{\boldsymbol{\theta}} + \sigma^2 I|}_{\text{complexity penalty}} - \frac{N}{2}\log(2\pi). \quad (36)$$

- ▶ An extremely powerful mechanism for kernel learning.



Aside: How Do We Build Models that Generalize?



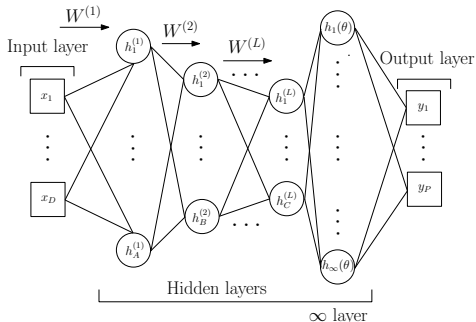
- ▶ Support: which datasets (hypotheses) are a priori possible.
- ▶ Inductive Biases: which datasets are a priori likely.

Want to make the *support* of our model as big as possible, with inductive biases which are calibrated to particular applications, so as to not rule out potential explanations of the data, while at the same time quickly learn from a finite amount of information on a particular application.

“How can Gaussian processes possibly replace neural networks? Have we thrown the baby out with the bathwater?” (MacKay, 1998)

Deep Kernel Learning

We combine the inductive biases of deep learning architectures with the non-parametric flexibility of Gaussian processes as part of a scalable *deep kernel learning* framework.



Deep Kernel Learning

Andrew Gordon Wilson*, Zhiting Hu*, Ruslan Salakhutdinov, and Eric P. Xing
Artificial Intelligence and Statistics (AISTATS), 2016.

Deep Kernel Learning

Starting from a base kernel $k(\mathbf{x}_i, \mathbf{x}_j | \boldsymbol{\theta})$ with hyperparameters $\boldsymbol{\theta}$, we transform the inputs (predictors) \mathbf{x} as

$$k(\mathbf{x}_i, \mathbf{x}_j | \boldsymbol{\theta}) \rightarrow k(g(\mathbf{x}_i, \mathbf{w}), g(\mathbf{x}_j, \mathbf{w}) | \boldsymbol{\theta}, \mathbf{w}), \quad (37)$$

where $g(\mathbf{x}, \mathbf{w})$ is a non-linear mapping given by a deep architecture, such as a deep convolutional network, parametrized by weights \mathbf{w} .

We use *spectral mixture* base kernels (Wilson, 2014):

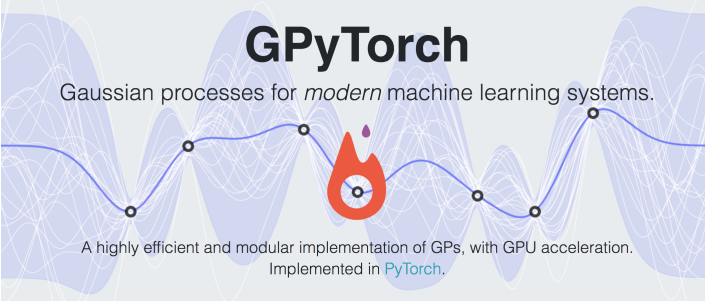
$$k_{\text{SM}}(\mathbf{x}, \mathbf{x}' | \boldsymbol{\theta}) = \sum_{q=1}^Q a_q \frac{|\Sigma_q|^{\frac{1}{2}}}{(2\pi)^{\frac{D}{2}}} \exp\left(-\frac{1}{2} \|\Sigma_q^{\frac{1}{2}}(\mathbf{x} - \mathbf{x}')\|^2\right) \cos\langle \mathbf{x} - \mathbf{x}', 2\pi\boldsymbol{\mu}_q \rangle. \quad (38)$$

Backprop to learn all parameters *jointly* through the GP marginal likelihood

Scalable Gaussian Processes

- ▶ Run Gaussian processes on **millions** of points in **seconds**, instead of thousands of points in hours.
- ▶ Outperforms stand-alone deep neural networks by learning **deep kernels**.
- ▶ Approach is based on kernel approximations which admit fast matrix vector multiplies (Wilson and Nickisch, 2015).
- ▶ Harmonizes with GPU acceleration.
- ▶ $\mathcal{O}(n)$ training and $\mathcal{O}(1)$ testing (instead of $\mathcal{O}(n^3)$ training and $\mathcal{O}(n^2)$ testing).
- ▶ **Implemented in our new library GPyTorch:**

<https://github.com/cornellius-gp/gpytorch>



GPyTorch

Gaussian processes for *modern* machine learning systems.

A highly efficient and modular implementation of GPs, with GPU acceleration.
Implemented in [PyTorch](#).

The image features a blue background with a wavy pattern of overlapping lines. A central red flame icon contains a white circle with a black dot. A blue line with several white circular markers passes through the flame icon.

Deep Kernel Learning Results

Datasets	n	d	RMSE						Runtime(s)		
			GP			DNN	DKL		DNN	DKL	
			RBF	SM	best		RBF	SM		RBF	SM
Gas	2,565	128	0.21±0.07	0.14±0.08	0.12±0.07	0.11±0.05	0.11±0.05	0.09±0.06	7.43	7.80	10.52
Skillcraft	3,338	19	1.26±3.14	0.25±0.02	0.25±0.02	0.25±0.00	0.25±0.00	0.25±0.00	15.79	15.91	17.08
SML	4,137	26	6.94±0.51	0.27±0.03	0.26±0.04	0.25±0.02	0.24±0.01	0.23±0.01	1.09	1.48	1.92
Parkinsons	5,875	20	3.94±1.31	0.00±0.00	0.00±0.00	0.31±0.04	0.29±0.04	0.29±0.04	3.21	3.44	6.49
Pumadyn	8,192	32	1.00±0.00	0.21±0.00	0.20±0.00	0.25±0.02	0.24±0.02	0.23±0.02	7.50	7.88	9.77
Pole/Tele	15,000	26	12.6±0.3	5.40±0.3	4.30±0.2	3.42±0.05	3.28±0.04	3.11±0.07	8.02	8.27	26.95
Elevators	16,599	18	0.12±0.00	0.090±0.001	0.089±0.002	0.099±0.001	0.084±0.002	0.084±0.002	8.91	9.16	11.77
Kin40k	40,000	8	0.34±0.01	0.19±0.02	0.06±0.00	0.11±0.01	0.05±0.00	0.03±0.01	19.82	20.73	24.99
Protein	45,730	9	1.64±1.66	0.50±0.02	0.47±0.01	0.49±0.01	0.46±0.01	0.43±0.01	142.8	154.8	144.2
KEGG	48,827	22	0.33±0.17	0.12±0.01	0.12±0.01	0.12±0.01	0.11±0.00	0.10±0.01	31.31	34.23	61.01
CTslice	53,500	385	7.13±0.11	2.21±0.06	0.59±0.07	0.41±0.06	0.36±0.01	0.34±0.02	36.38	44.28	80.44
KEGGU	63,608	27	0.29±0.12	0.12±0.00	0.12±0.00	0.12±0.00	0.11±0.00	0.11±0.00	39.54	42.97	41.05
3Droad	434,874	3	12.86±0.09	10.34±0.19	9.90±0.10	7.36±0.07	6.91±0.04	6.91±0.04	238.7	256.1	292.2
Song	515,345	90	0.55±0.00	0.46±0.00	0.45±0.00	0.45±0.02	0.44±0.00	0.43±0.01	517.7	538.5	589.8
Buzz	583,250	77	0.88±0.01	0.51±0.01	0.51±0.01	0.49±0.00	0.48±0.00	0.46±0.01	486.4	523.3	769.7
Electric	2,049,280	11	0.230±0.000	0.053±0.000	0.053±0.000	0.058±0.002	0.050±0.002	0.048±0.002	3458	3542	4881

Similar runtime but improved performance over stand-alone DNNs and scalable kernel learning on a wide range of benchmarks

Face Orientation Extraction

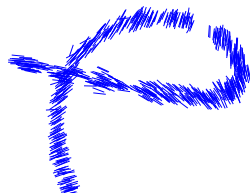


Figure: **Left:** Randomly sampled examples of the training and test data. **Right:** The two dimensional outputs of the convolutional network on a set of test cases. Each point is shown using a line segment that has the same orientation as the input face.

Learning Flexible Non-Euclidean Similarity Metrics

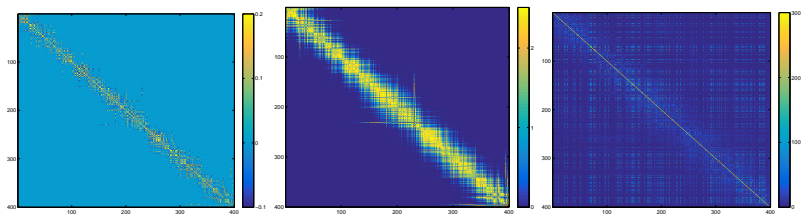


Figure: **Left:** The induced covariance matrix using DKL-SM kernel on a set of test cases, where the test samples are ordered according to the *orientations* of the input faces. **Middle:** The respective covariance matrix using DKL-RBF kernel. **Right:** The respective covariance matrix using regular RBF kernel. The models are trained with $n = 12,000$. We set $Q = 4$ for the SM base kernel.

Step Function

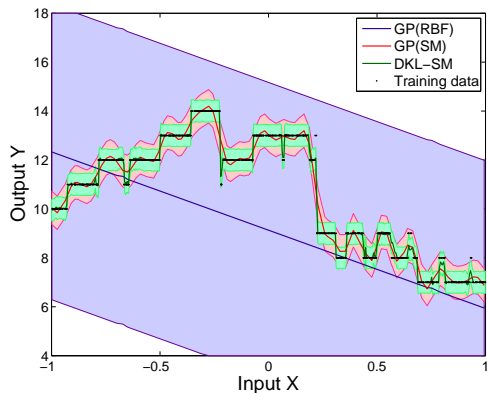
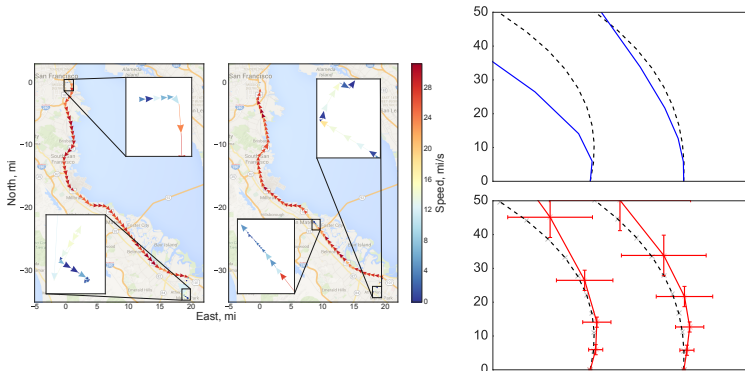


Figure: Recovering a step function. We show the predictive mean and 95% of the predictive probability mass for regular GPs with RBF and SM kernels, and DKL with SM base kernel. We set $Q = 4$ for SM kernels.

LSTM Kernels

- ▶ We derive kernels which have recurrent LSTM inductive biases, and apply to autonomous vehicles, where predictive uncertainty is critical.

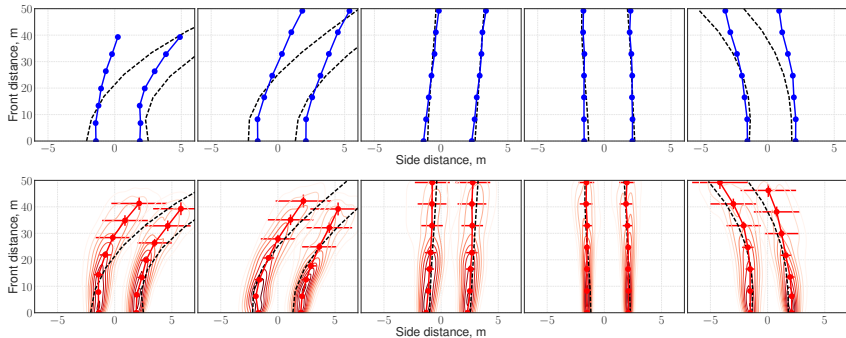


Learning Scalable Deep Kernels with Recurrent Structure

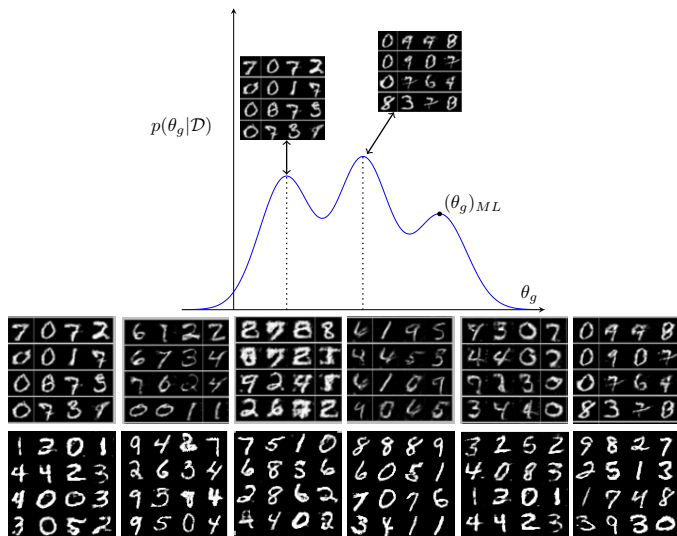
M. Al-Shedivat, A. G. Wilson, Y. Saatchi, Z. Hu, E. P. Xing

Journal of Machine Learning Research (JMLR), 2017

GP-LSTM Predictive Distributions

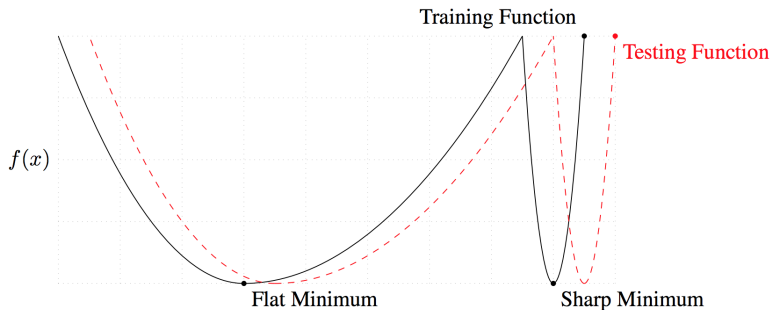


The Bayesian GAN



Wilson and Saatchi (NIPS 2017)

Wide Optima Generalize Better



Keskar et. al (2017)

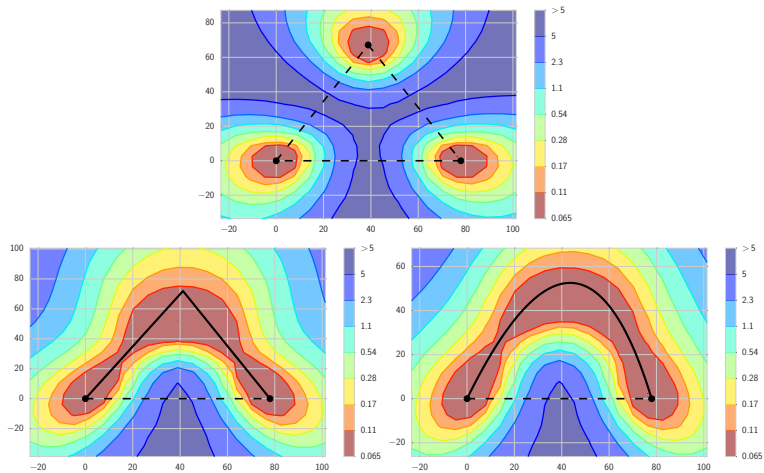
- **Bayesian integration will give very different predictions in deep learning especially!**

Loss Surfaces in Deep Learning

- (1) *Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs*
- (2) *Averaging Weights Leads to Wider Optima and Better Generalization*
- (3) *Consistency Based Semi-Supervised Learning with Weight Averaging*

- ▶ Local optima are connected along simple curves
- ▶ SGD does not converge to broad optima in all directions
- ▶ Averaging weights along SGD trajectories with constant or cyclical learning rates leads to much faster convergence and solutions that generalize better.
- ▶ Can approximate ensembles and Bayesian model averages with a single model.

Mode Connectivity



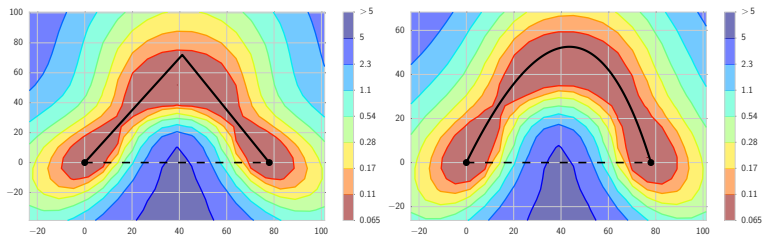
Example Parametrizations

Polygonal Chain:

$$\phi_{\theta}(t) = \begin{cases} 2(t\theta + (0.5 - t)\hat{w}_1), & 0 \leq t \leq 0.5 \\ 2((t - 0.5)\hat{w}_2 + (1 - t)\theta), & 0.5 \leq t \leq 1. \end{cases} \quad (39)$$

Bezier Curve:

$$\phi_{\theta}(t) = (1 - t)^2 \hat{w}_1 + 2t(1 - t)\theta + t^2 \hat{w}_2, \quad 0 \leq t \leq 1. \quad (40)$$



Connection Procedure with Tractable Loss

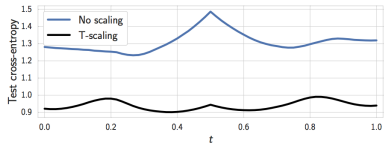
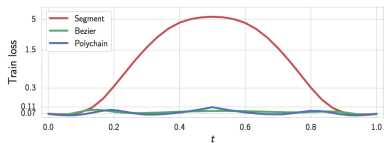
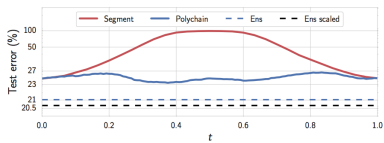
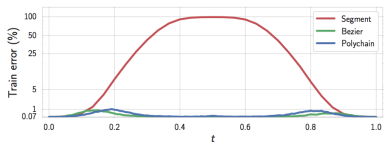
We propose the computationally tractable loss:

$$\ell(\theta) = \int_0^1 \mathcal{L}(\phi_\theta(t)) dt = \mathbb{E}_{t \sim U(0,1)} \mathcal{L}(\phi_\theta(t)) \quad (41)$$

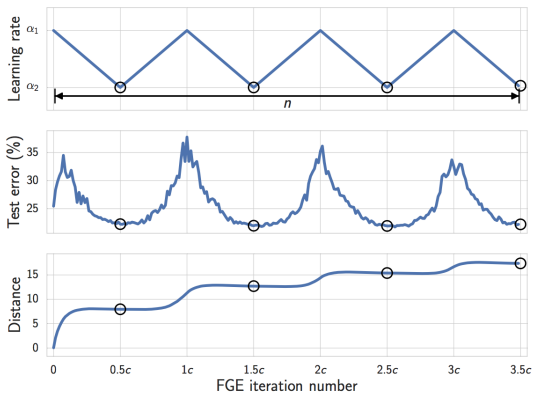
At each iteration we sample $t \in [0, 1]$ and make a gradient step for θ with respect to $\mathcal{L}(\phi_\theta(t))$:

$$\mathbb{E}_{t \sim U(0,1)} \nabla_\theta \mathcal{L}(\phi_\theta(t)) = \nabla_\theta \mathbb{E}_{t \sim U(0,1)} \mathcal{L}(\phi_\theta(t)) = \nabla_\theta \ell(\theta).$$

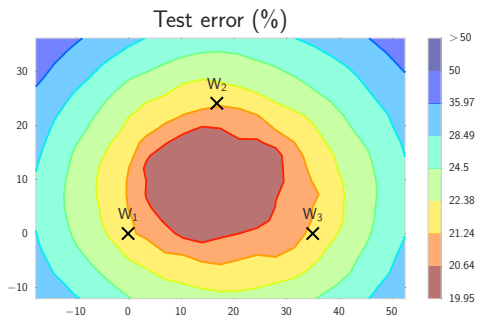
Curve Ensembling



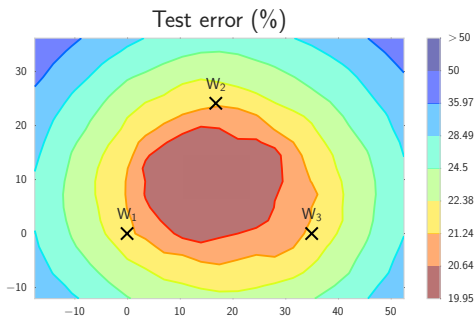
Fast Geometric Ensembling



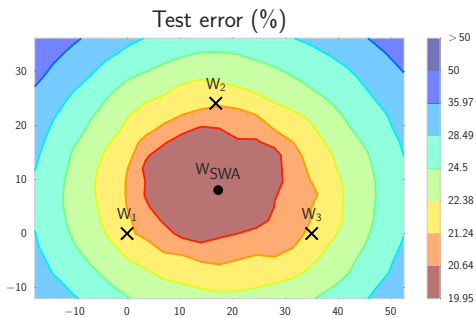
Trajectory of SGD



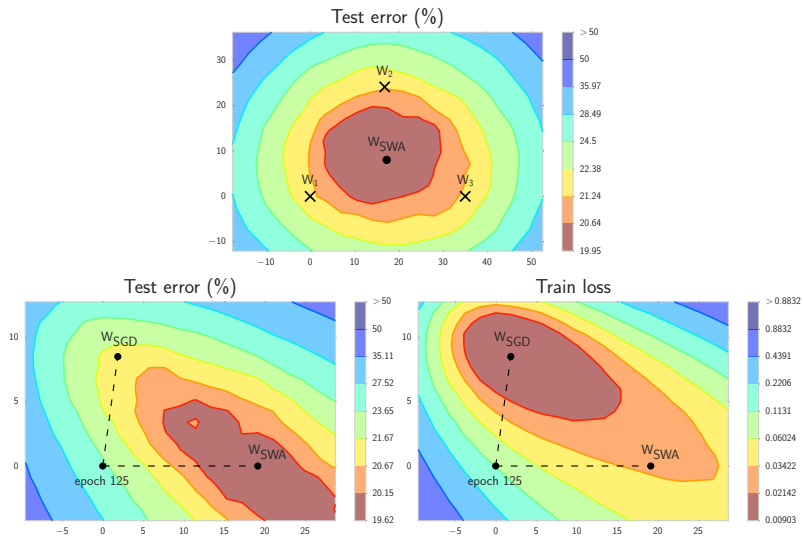
Trajectory of SGD



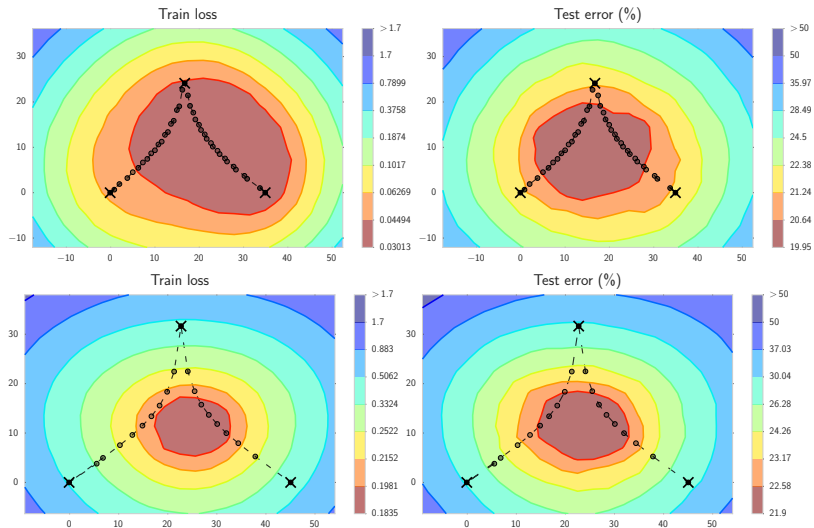
Trajectory of SGD



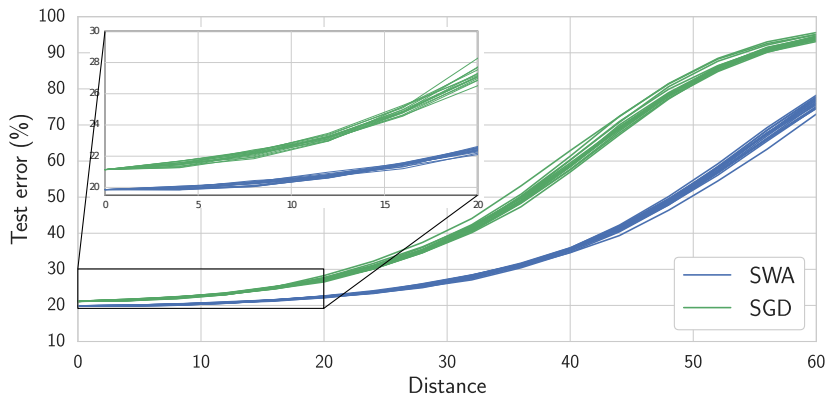
Trajectory of SGD



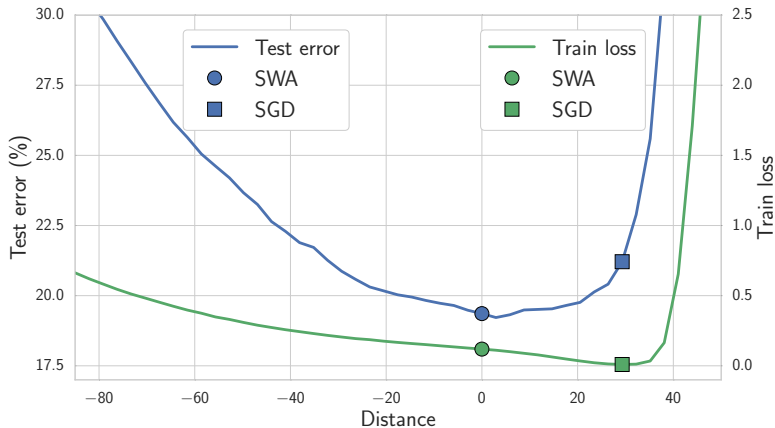
Trajectory of SGD



Following Random Paths



Path from w_{SWA} to w_{SGD}



Because the points sampled from an FGE ensemble take small steps in weight space *by design*, we can do a linearization analysis to show that

$$f(w_{\text{SWA}}) \approx \frac{1}{n} \sum f(w_i)$$

SWA Results, CIFAR

Table 1: Accuracies (%) of SWA, SGD and FGE methods on CIFAR-100 and CIFAR-10 datasets for different training budgets. Accuracies for FGE were taken from [Garipov et al., 2018].

DNN (Budget)	SGD	FGE (1 Budget)	SWA		
			1 Budget	1.25 Budgets	1.5 Budgets
CIFAR-100					
VGG-16 (200)	72.55 \pm 0.10	74.26	73.91 \pm 0.12	74.17 \pm 0.15	74.27 \pm 0.25
ResNet-110 (150)	78.49 \pm 0.36	79.84	79.77 \pm 0.17	80.18 \pm 0.23	80.35 \pm 0.16
WRN-28-10 (200)	80.82 \pm 0.23	82.27	81.46 \pm 0.23	81.91 \pm 0.27	82.15 \pm 0.27
PyramidNet-272 (300)	83.41 \pm 0.21	–	–	83.93 \pm 0.18	84.16 \pm 0.15
CIFAR-10					
VGG-16 (200)	93.25 \pm 0.16	93.52	93.59 \pm 0.16	93.70 \pm 0.22	93.64 \pm 0.18
ResNet-110 (150)	95.28 \pm 0.10	95.45	95.56 \pm 0.11	95.77 \pm 0.04	95.83 \pm 0.03
WRN-28-10 (200)	96.18 \pm 0.11	96.36	96.45 \pm 0.11	96.64 \pm 0.08	96.79 \pm 0.05
ShakeShake-2x64d (1800)	96.93 \pm 0.10	–	–	97.16 \pm 0.10	97.12 \pm 0.06

SWA Results, ImageNet (Top-1 Error Rate)

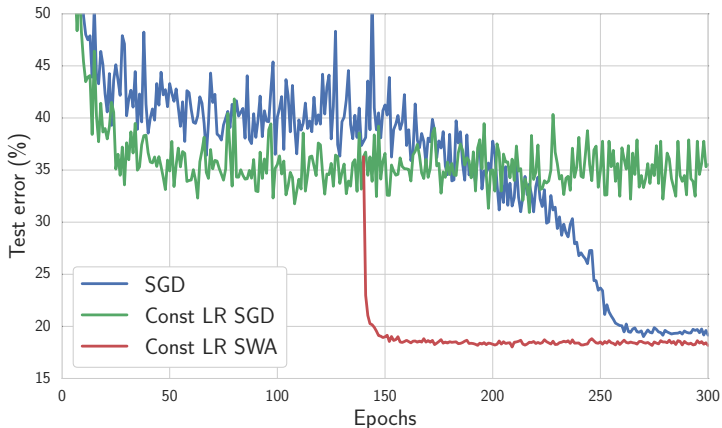
DNN	SGD	SWA	
		5 epochs	10 epochs
ResNet-50	76.15	76.83 ± 0.01	76.97 ± 0.05
ResNet-152	78.31	78.82 ± 0.01	78.94 ± 0.07
DenseNet-161	77.65	78.26 ± 0.09	78.44 ± 0.06

Sampling from a High Dimensional Gaussian



SGD (with constant LR) proposals are on the surface of a hypersphere. Averaging lets us go inside the sphere to a point of higher density.

High Constant LR



Side observation: Averaging bad models does not give good solutions. Averaging bad weights can give great solutions.

Conclusions

- ▶ Bayesian deep learning can provide better predictions and uncertainty estimates
- ▶ Computation is a key challenge. We address this issue through developing numerical linear algebra approaches.
- ▶ Developing fast deterministic (e.g., variational) approximations and stochastic MCMC algorithms will be important too.
- ▶ We can better understand model construction, and generalization, by taking a *function space* approach.
- ▶ We can be inspired by Bayesian methods to develop optimization procedures that generalize better.
- ▶ Code for the approaches discussed here is available at:
<https://people.orie.cornell.edu/andrew/code>

Appendix

Deriving the RBF Kernel

$$f(x) = \sum_{i=1}^J w_i \phi_i(x), \quad w_i \sim \mathcal{N}\left(0, \frac{\sigma^2}{J}\right), \quad \phi_i(x) = \exp\left(-\frac{(x - c_i)^2}{2\ell^2}\right) \quad (42)$$

$$\therefore k(x, x') = \frac{\sigma^2}{J} \sum_{i=1}^J \phi_i(x) \phi_i(x') \quad (43)$$

- ▶ Let $c_J = \log J$, $c_1 = -\log J$, and $c_{i+1} - c_i = \Delta c = 2\frac{\log J}{J}$, and $J \rightarrow \infty$, the kernel in Eq. (43) becomes a Riemann sum:

$$k(x, x') = \lim_{J \rightarrow \infty} \frac{\sigma^2}{J} \sum_{i=1}^J \phi_i(x) \phi_i(x') = \int_{c_0}^{c_\infty} \phi_c(x) \phi_c(x') dc \quad (44)$$

- ▶ By setting $c_0 = -\infty$ and $c_\infty = \infty$, we spread the infinitely many basis functions across the whole real line, each a distance $\Delta c \rightarrow 0$ apart:

$$k(x, x') = \int_{-\infty}^{\infty} \exp\left(-\frac{(x - c)^2}{2\ell^2}\right) \exp\left(-\frac{(x' - c)^2}{2\ell^2}\right) dc \quad (45)$$

$$= \sqrt{\pi} \ell \sigma^2 \exp\left(-\frac{(x - x')^2}{2(\sqrt{2}\ell)^2}\right). \quad (46)$$

Deriving the RBF Kernel

- ▶ It is remarkable we can work with infinitely many basis functions with finite amounts of computation using the *kernel trick* – replacing inner products of basis functions with kernels.
- ▶ The RBF kernel, also known as the Gaussian or squared exponential kernel, is by far the most popular kernel. $k_{\text{RBF}}(x, x') = a^2 \exp(-\frac{\|x-x'\|^2}{2\ell^2})$.
- ▶ Functions drawn from a GP with an RBF kernel are infinitely differentiable. For this reason, the RBF kernel is accused of being overly smooth and unrealistic. Nonetheless it has nice theoretical properties...

References

- ▶ A.G. Wilson and H. Nickisch. Kernel interpolation for scalable structured Gaussian processes (KISS-GP). *International Conference on Machine Learning (ICML)*, 2015.
- ▶ P. Izmailov*, D. Podoprikin*, T. Garipov*, D. Vetrov, A.G. Wilson. Averaging Weights Leads to Wider Optima and Better Generalization, *Uncertainty in Artificial Intelligence (UAI)*, 2018.
- ▶ T. Garipov*, P. Izmailov*, D. Podoprikin*, D. Vetrov, A.G. Wilson. Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs, 2018.
- ▶ R. Neal. Bayesian Learning for Neural Networks. PhD Thesis, 1996.
- ▶ D. MacKay. Introduction to Gaussian Processes. *Neural Networks and Machine Learning*, 1998.
- ▶ D. MacKay. Information Theory, Inference, and Learning Algorithms, 2003.
- ▶ D. MacKay. Bayesian Interpolation, 1992.
- ▶ M. Welling and Y.W. Teh. Bayesian learning via stochastic gradient langevin dynamics. *International Conference on Machine Learning (ICML)*, 2011.

References

- ▶ T. Chen, E. Fox, and C. Guestrin. Stochastic gradient Hamiltonian Monte Carlo. In International Conference on Machine Learning (ICML), 2014.
- ▶ J. Gardner, G. Pleiss, K. Weinberger, A.G. Wilson. *GPYtorch: Scalable and Modular Gaussian Processes in PyTorch*: <https://github.com/cornellius-gp/gpytorch>.
- ▶ Codebase for group projects: <https://people.orie.cornell.edu/andrew/code>.
- ▶ B. Athiwaratkun, M. Finzi, P. Izmailov, A.G. Wilson. Improving Consistency-Based Semi-Supervised Learning with Weight Averaging.
- ▶ A.G. Wilson*, Z. Hu*, R. Salakhutdinov, and E.P. Xing. Deep kernel learning. *Artificial Intelligence and Statistics (AISTATS)*, 2016.
- ▶ A.G. Wilson*, Z. Hu*, R. Salakhutdinov, and E.P. Xing. Stochastic Variational Deep Kernel Learning. *Neural Information Processing Systems (NIPS)*, 2016.
- ▶ M. Al-Shedivat, A.G. Wilson, Y. Saatchi, Z. Hu, and E.P. Xing. Learning Scalable Deep Kernels with Recurrent Structure. *Journal of Machine Learning Research (JMLR)*, 2017.
- ▶ C. E. Rasmussen and C.K.I. Williams. Gaussian Processes for Machine Learning. Cambridge Press, 2006.
- ▶ N.S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, P. Tang. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. ICLR, 2017.